arXiv:2602.04889v1 [cs.FL] 26 Jan 2026

模板化组装理论：以块压缩模板扩展规范组装指数

Piotr Masierak

Łukaszyk 专利律师事务所，ul. Głowackiego 8, 40-052 卡托维兹，波兰
pmasierak@patent.pl

2026年1月26日

摘要

由 Cronin 及其合作者发展的组装理论为对象赋予一个组装指数：从一组指定的基本构件出发，允许复用中间组件，构建该对象至少一个副本所需的二元连接操作的最小次数。对于有限字母表上的字符串，规范组装指数可在自由半群（Σ^+,·）中定义，采用通用二元拼接及“无废料”条件，且其精确计算已被证明为 NP 完全。本文提出对基于字符串的规范表述的一种扩展，以模板化组装步骤增强纯拼接。中间对象可包含一个特殊的通配符 $*$，用以表示可压缩块。模板仅限于目标字符串的块压缩子串，并可通过将先前组装的基序插入一个或多个通配符位置来实例化，且可能并行进行。由此得到一种新的复杂度度量——模板化组装指数，它严格推广了规范指数，同时保留其操作性特征。我们对该模型进行形式化，阐明其与规范组装指数以及最小文法问题等经典问题的关系，并讨论确定模板化组装指数的计算复杂性。最后，我们概述其在序列分析、模块化检测和生物特征设计中的潜在应用。

关键词： 组装理论、组装指数、信息论、复杂性度量

1 引言

组装理论（AT）旨在通过计算从简单组件构建对象所需的最少组合步骤，来量化对象的历史偶然性和结构复杂性；这一过程基于一组指定的允许的连接，并允许重复使用先前组装好的子结构 [1, 2, 3]。在基于字符串的形式中，我们考虑有限字母表 Σ 上的自由半群（Σ^+，·），其中二元拼接是唯一的连接操作。字符串 $w \in \Sigma^+$ 的规范组装指数，定义为生成至少一个 w副本的组装计划的最短长度；其中只能使用 w 的子串作为中间对象，且每次连接的成本均为单位成本。这种规范的“无废料”变体，为更具操作性的领域特定版本的组装理论提供了普适下界。

在 [4] 中提出了这一猜想，并在 [5] 中证明：计算给定字符串的组装指数——甚至判断该指数是否不超过给定上界 k——都是 NP 完全的；这一结论通过从最小文法问题归约而来，并基于组装计划与直线程序之间的对应关系。从结构角度来看，规范指数对

子串的层级复用很敏感，尤其是那些可以组装一次、再在后续连接中复制的重复基序。

然而，许多自然系统展现出某些模板式或基于模式的组装形式，单纯的拼接无法充分描述这些形式。例如，固定的分子骨架上可能有多个位点连接着同一种官能团；又如，调控 DNA 序列中可能嵌有反复出现、但局部上下文各异的结合基序。从字符串组装的角度看，这些现象类似于使用模板：模板是带有空缺（通配符位置）的模式，随后由先前组装好的基序填充这些空缺，而且可能同时填充多个位置。

本文的核心目标是以一种忠于组装理论操作理念的方式，将这一直觉形式化。我们提出了规范的基于字符串的模型的一种新变体，其中间对象可以包含一个特殊符号 $*$ ，表示任意非空字符串的占位符。关键的是，我们将模板限定为由目标字符串的块压缩子串构成：粗略地说，模板是从目标 w 中取出子串 u ，再将其中若干连续块 u 替换为 $*$，从而将整块压缩为单个通配符而得到的。如今，组装操作不仅包括拼接，还包括模板实例化步骤：在单个步骤中，将先前可用的基序 x 替换模板 T中一个或所有出现的 $*$ 。

由此得到一种新的复杂度度量，即模板化组装指数 TAI(w)。按照设计，规范指数 ASI(w) 作为从不使用通配符模板的特例出现，并且我们证明，对所有 w 都有 TAI(w) $\leq$ ASI(w)。与此同时，对于表现出非局部、模板化基序复用的字符串，TAI(w) 可以严格小于 ASI(w)，从而使 TAI 对结构的另一侧面——我们称之为模板化模块性——更为敏感。

2 背景：规范的基于字符串的组装理论

我们简要回顾组装理论的规范的基于字符串的表述，并固定记号。设 Σ 为一个有限字母表，并设 Σ^+ 表示 Σ 上所有非空字符串的集合。我们考虑以拼接为二元操作的自由半群 $(\Sigma^+,\cdot)$。

定义 2.1. 设 $w \in \Sigma^+$。 w 的规范组装空间是如下二元组

$$\mathcal{A}(w) = (O, R),$$

其中：

- $O \subseteq \Sigma^+$ 是对象集合，取为 w 的所有非空子串构成的集合 Sub(w)；
- R 是形式如下的二元连接规则集合

$$x \circ y \to xy$$

对于所有 x， $y \in O$ ，满足 $xy \in O$。

通俗地说， O 恰好包含目标 w的所有子串，并且只要两个子串x， y 拼接后 xy 仍是 w的子串，我们就可以在一个组装步骤中将它们连接起来。因此，中间对象受到无废弃条件的约束：任何组装计划中出现的每个对象都是目标 w的子串。

定义 **n 2.2.** 设 $w \in \Sigma^+$ 和$\mathcal{A}(w) = (O,\ R)$如上。 w 的组装计划是一个序列 e

$$x_1, x_2, \ldots, x_m$$

对象的定义，其中 O 满足：

(a) 每个 x_i 要么是 Σ 中的一个字母，要么由两个较早出现的对象通过 R中的一条规则得到；也就是说，存在 $j, k < i$ ，使得$x_i = x_j x_k$ 且 $x_j x_k \in O$；

(b) x_i中至少有一个位置出现 w 。

组装计划的长度是复合步骤的数量，也就是满足以下条件的索引 i的数量： x_i 通过 R中的某条连接规则得到。 w的规范组装指数记为 ASI(w)，是 w的组装计划的最小长度。

这种表述体现了这样的直觉：我们从单体构件（单个字母）开始，在每一步中连接两个此前可用的对象，每次连接都支付单位成本，直到至少一次构造出目标 w 。关键的是，其中内置了重复使用原则：一旦对象 x 出现在组装计划中，之后就可以任意多次用于连接，实际上无需额外的边际成本。

3 模板化组装空间

现在，我们介绍规范组装空间模型的一种模板化扩展。在本节中，我们固定一个有限字母表 Σ 和一个特殊的通配符 $* \notin \Sigma$。

直观而言，模板是将目标词的一个子串中的整个连续块压缩为单个通配符 $*$后得到的。之后将在通配符位置插入先前组装好的基序，以实例化这些位置。

定义 3.1。 设 $w \in \Sigma^+$。若存在索引 $0 \le i \le j < |w|$ 和一个划分，则称字符串 $T \in (\Sigma \cup \{*\})^+$ 是 w 的块压缩模板

$$0 = b_0 < b_1 < \cdots < b_k = L$$

of t长度为 $L = j - i + 1$ 的子串 $u = w[i..j]$ 的位置划分为连续块 cks

$$S_r = u[b_r..b_{r+1} - 1] \in \Sigma^+, \quad r = 0, \ldots, k-1,$$

以及位向量 $F \in \{0,1\}^k$ ，满足：

- F 既不全为零，也不全为一；
- T 通过将每个块 S_r 替换为

$$T_r = \begin{cases} S_r & \text{if } F[r] = 0, \\ * & \text{if } F[r] = 1, \end{cases}$$

并令 $T = T_0 T_1 \cdots T_{k-1}$。

我们将 w 的所有块压缩模板的集合记作 $\pi(w)$。

请注意，模板并非假定为独立于输入而给定：对于固定的目标字符串 w ，唯一可容许的模板是 $\pi(w)$ 中的模板（即通过对 w 的子串进行分块压缩而挖掘得到），并且只有在模板

已通过 R_{concat} 中的规则从单体显式构造出来之后，才能在组装计划中使用（可能会重复利用先前组装好的对象）。

因此， T 是由 w 的某个连续子串 u 通过将一个或多个完整的块压缩为单个通配符而得到的；至少有一个块保持为字面形式，且至少有一个块被压缩。通配符 $*$ 可以理解为一个“空洞”，用于标记稍后可插入任意非空字符串的区域。

请注意，块边界是任意的，因此压缩块的长度可以为1。特别地，模板可以编码“单点”可变性：一个通配符位置可以对应于底层子串中的单个符号，而用 Σ中的不同单体对其进行实例化，便可捕捉仅在特定位置上有所不同的模式。

注 3.2. 将限制限定为对子串进行块压缩，强制实现了无废弃条件的模板化版本：每个模板 T 都通过其字面片段锚定在 w 中，这些字面片段的出现顺序与在 w中相同。特别地，用本身作为 w 的子串出现的可压缩块对 T 进行的任何实例化，都会再次得到 w的一个子串。

我们现在定义与目标字符串 w 相关联的模板化组装空间中的对象和操作。

定义 3.3. 设 $w \in \Sigma^+$。 w 的模板化组装空间是三元组

$$\mathcal{A}_*(w) = (O_*, R_{\text{concat}}, R_{\text{temp}}),$$

其中

- O_* 是对象的集合，定义为

  $$O_* = \text{Sub}(w) \cup \mathcal{T}(w).$$

  因此，每个对象要么是 w 的子串（不含通配符），要么是 w 的块压缩模板（至少包含一个通配符和一个字面片段）。

- $R_{\text{拼接}}$ 是拼接规则的集合

  $$x \circ y \to xy$$

  对于所有满足 $xy \in O_*$ 的 $x, y \in O_*$。

- R_{temp} 是模板实例化规则的集合。对于模板 $T \in \mathcal{T}(w)$，令 $P(T)$ 表示 T 中通配符位置的集合，即

  $$P(T) = \{\, p : T[p] = * \,\}.$$

  对于每个非空子集 $S \subseteq P(T)$ 和每个满足 $T[S \mapsto u] \in O_*$ 的对象 $u \in O_*$，我们有一条规则

  $$T \bullet_S u \to T[S \mapsto u],$$

  其中， $T[S \mapsto u]$ 表示由 T 在 S 中每个通配符所在位置替换为 u ，并保持所有其他位置不变后得到的字符串。特别地，如果 $S =\{p\}$是一个单元素集合，则我们得到在位置 p 进行的单星替换；如果 $S = P(T)$，则我们得到完全并行替换，即同时将 T 中的所有通配符替换为 u 。与之前一样，我们明确要求 $T[S \mapsto u] \in O_*$；只要 u是 w的子串，这一条件就会自动满足，但一般情况下，我们将其作为无废弃条件的一部分加以规定。

备注 3.4。 $R_{\text{concat}} \cup R_{\text{temp}}$ 中的每次规则应用成本均为单位成本。

模板并非外部原语。组装w 时唯一可以使用的模板是$\mathcal{T}(w)$中的那些，即从目标字符串的子串中挖掘出的模板（见上文定义）。此外，模板 $T \in \mathcal{T}(w)$只有在先通过 R_{concat} 中的常规二元连接规则、由单体（ Σ 中的字母以及通配符 $*$）构建完成之后，才能出现在组装计划中。特别地，模板的构建可以复用同一计划中更早已经构建的任何对象（包括子串以及先前组装好的模板）。

因此，“构建”一个模板的总成本恰好是其组装中所用拼接步骤的数量，而此后每一次来自 R_{temp} 的模板实例化步骤都计为一次独立的单位代价操作。

注记 3.5。 实例化规则 $T \bullet_S u \to T[S \mapsto u]$ 将单星号替换（当 $|S| = 1$时）和完全并行替换（当 $S = P(T)$时）都作为特例纳入。在此意义上，当前的模板化模型同时涵盖了单星号变体与全星号变体。

因此，组装操作现在既包括对象的普通拼接，也包括将先前组装的基序模板化插入块压缩模板中。需要强调的是，模板本身并不是通过特殊规则引入的：它们通过对子串进行分块压缩，由 w静态确定，并且从一开始就是 O_* 的一部分。所有动态操作都是二元的，且单位成本为 1，这与规范模型一致。

接下来，我们将在模板化设定中定义组装计划及其相关的复杂度度量。

定义 3.6。 设 $w \in \Sigma^+$ 和 $\mathcal{A}_*(w) = (O_*,\ R_{\text{concat}},\ R_{\text{temp}})$。对于$w$ ，模板化组装方案是对象的有限序列

$$x_1, x_2, \ldots, x_m$$

其中对象属于 O_* ，并满足：

(a) 每个 x_i 要么是单体（ Σ 中的单个字母或通配符 $*$），要么是通过对先前对象应用 $R_{\text{concat}} \cup R_{\text{temp}}$ 中的规则得到的；

(b) w 至少有一次出现在 x_i中。

组装计划的长度是复合步骤的数量，即通过规则在 $R_{\text{concat}} \cup R_{\text{temp}}$ 中得到 x_i 的索引 i 的数量。

现在我们可以定义模板化组装指数了。

定义 3.7。 设 $w \in \Sigma^+$。 w 的模板化组装指数记为 $\text{TAI}(w)$，它是在 $\mathcal{A}_*(w)$ 中针对 w 的所有模板化组装方案中，长度（复合步骤的数量）最小者。

根据构造，我们始终可以忽略模板，只考虑子串和拼接，从而得到一个标准组装计划。

命题 3.8。对于所有 $w \in \Sigma^+$ ，我们有

$$\text{TAI}(w) \ \le\ \text{ASI}(w).$$

此外，如果 w 不允许使用 R_{temp} 中的规则制定任何模板化组装方案，那么 $\text{TAI}(w) = \text{ASI}(w)$。

概述。 w中的任何标准组装计划也是$\mathcal{A}(w)$ 中的有效模板化组装方案$\mathcal{A}_*(w)$：子串集合 Sub(w) 包含于 O_* 中，并且所有规范连接规则都出现在 R_{concat} 中。因此，作为在更大组装计划集合上取最小值的 TAI(w) 不可能超过 ASI(w)。

如果在任何最优模板化组装计划中，都无法应用 R_{temp} 中的模板规则而不违反对象约束（O_*），那么任何最优模板化组装计划都只使用拼接规则；将范围限制为子串即可恢复出一个长度相同的标准组装计划，因此 TAI(w) =ASI(w)。 □

实际上，对于包含嵌入在非局部上下文中的重复基序、且这些上下文可由块压缩模板描述的字符串，TAI(w) 可以严格小于 ASI(w)。我们将在下一小节中说明这一现象。

备注 3.9。 更一般地，设 O 为一个固定的对象全集， $R_1 \subseteq R_2$ 为 O上的两个组装规则集合。对于 $i \in \{1, 2\}$，令 ASI$_i$(w) 表示仅允许使用 R_i 中的规则时， w 的组装计划的最小长度。那么对于每个 w ，都有 ASI$_2$(w) ≤ASI$_1$(w)，因为任何 R_1-计划也是有效的 R_2-计划。特别地，通过从 R_{temp} 中移除某些实例化规则而得到的任何限制性更强的模板化模型，都会产生一个满足下式的指标 TAI′(w)

$$\text{TAI}(w) \;\le\; \text{TAI}'(w) \;\le\; \text{ASI}(w).$$

因此，当前的 TAI(w) 可视为一族规范指标中最小的成员，这一族指标通过改变模板规则集的强度而得到。

3.1 Ill说明性示例：选择性模板挖掘以及通配符何时有帮助 p

当前模型的一个关键点是，模板是从目标字符串 w中挖掘出来的：可容许模板恰好是$\mathcal{T}(w)$中的块压缩模板。TAI(w)指标的定义，是对$\mathcal{A}_*(w)$中的所有模板化组装计划进行最小化。因此，只有在模板有益时才会使用模板：如果没有任何采用 R_{temp} 中的规则的计划优于最佳的纯拼接计划，那么最优计划就不会使用模板，由此得到 TAI(w) =ASI(w)（上文命题）。

要了解通配符机制何时能够减少步骤数，可以将两个问题分开考察：（i）组装模板骨架的成本（通过 R_{concat}计数）；（ii）实例化通配符位置的成本（通过 R_{temp}计数）。

- **单次使用模板**。 如果模板 T 仅被组装并实例化一次，那么引入它通常并无帮助：由于 T 不是原始对象，因此必须付出代价（通过$R_{\text{拼接}}$）来构造其骨架，然后还要在 $R_{\text{模板}}$中额外支付一次单位成本实例化步骤。具体而言，当 $|P(T)| = 1$（即 T只有一个通配符位置）时，实例化步骤无法在多个位置之间实现并行性；因此，与纯拼接式组装相比，优势并非必然存在，而取决于组装计划中周围上下文是否会在其他地方复用。

- **通过复用摊销**。 一旦同一个组装骨架在组装计划中被多次实例化（可能使用不同的填充符），模板就可能带来益处。在这种情况下，构造 T 的一次性成本可以分摊到多次实例化中。对于 $|P(T)| = 1$，这种效果也可能出现，但前提是反复使用由 T表示的同一上下文“框架”。

- **多个位置的并行填充**。即使对于固定的目标 w，单个实例化规则也可以一次填充多个通配符位置（选择 $S \subseteq P(T)$）。因此，带有

$|P(T)| \geq 2$引入了一种真正全新的机制： R_{temp}中的一次操作可以同时实现模体的多次出现，这是相较于纯粹的拼接式组装而言潜在步数节省的主要来源。

作为示意性示例，考虑如下形式的字符串

$$w \;=\; P\,S, \qquad P = a_0\,u\,a_1\,u\,a_2\cdots a_{m-1}\,u\,a_m, \qquad S = a_0\,v\,a_1\,v\,a_2\cdots a_{m-1}\,v\,a_m,$$

其中 a_0，……， $a_m \in \Sigma^+$ 是固定的“锚点”，而 u， $v \in \Sigma^+$ 是两个模体。模板骨架 n

$$T \;=\; a_0 \,*\, a_1 \,*\, a_2\cdots a_{m-1} \,*\, a_m$$

是 $\pi(w)$ 的一个元素（它通过压缩模体块的每次出现，从 w 中挖掘得到），构建 T 一次后，我们可以将其实例化两次，从而得到 P 和 S。当锚点上下文较大且通配符位置至少有两个时，与仅通过拼接分别组装 P 和 S 相比，这可能产生更短的总体方案。下一小节中的示例提供了这一分离机制的具体实例。

3.2 一个 worked example：分离ASI 和 TAI

考虑字符串

$$w = 1122112211000110011 0011.$$

现在，我们在字母表 $\Sigma = \{0,1,2\}$ 上给出一个具体的 worked example，说明对于同一个目标字符串，模板化组装指数 TAI(w) 可以严格小于规范组装指数 ASI(w)。

规范组装计划（显示 $ASI(w) = 12$）。 在规范组装空间 $\mathcal{A}(w)$ 中，我们仅考虑 w 的子串和二元拼接。 w 的一种可能组装计划如下（每行代表一个单位代价步骤）：

$$\begin{aligned}
00 &= 0 + 0,\\
11 &= 1 + 1,\\
22 &= 2 + 2,\\
1100 &= 11 + 00,\\
2211 &= 22 + 11,\\
22110 &= 2211 + 0,\\
001100 &= 00 + 1100,\\
110011 &= 1100 + 11,\\
112211 &= 11 + 2211,\\
11221122110 &= 112211 + 22110,\\
001100110011 &= 001100 + 110011,\\
1122112211000110011 0011 &= 11221122110 + 001100110011.
\end{aligned}$$

这给出了一个长度为 12 的标准组装计划。我们已通过计算验证，不存在长度不超过 11 的标准组装计划。因此

$$ASI(w) = 12.$$

模板化组装方案和 TAI(w)。 为便于阅读，我们将字符串 w 分解为三个连续块

$$w = P\,M\,S,$$

其中

$$P = \texttt{1122112211}, \qquad M = \texttt{000}, \qquad S = \texttt{1100110011}.$$

P 和 S 共享相同的“锚点”骨架

$$\texttt{11}\,\square\,\texttt{11}\,\square\,\texttt{11},$$

其中占位符 $\square$ 在 P 中具体表现为 22，在 S中则表现为 00。这正是带有通配符 $*$ 的块压缩模板能够体现非局部模块性的情形。因此，在模板化组装空间$\mathcal{A}_*(w)$中，我们还可以使用通配符 $*$ 和块压缩模板

$$T = \texttt{11*11*11},$$

该模板是将 w 中的子串 1100110011 进行块压缩后得到的，其中三个锚点块 11 由两个通配符隔开。从单体开始，我们可以通过以下步骤构建 T：

$$\begin{aligned}
\texttt{11} &= \texttt{1} \circ \texttt{1},\\
\texttt{11*} &= \texttt{11} \circ *,\\
\texttt{11*11*} &= \texttt{11*} \circ \texttt{11*},\\
\texttt{11*11*11} &= \texttt{11*11*} \circ \texttt{11}.
\end{aligned}$$

我们还构建了基序 22 和 00：

$$\begin{aligned}
\texttt{22} &= \texttt{2} \circ \texttt{2},\\
\texttt{00} &= \texttt{0} \circ \texttt{0}.
\end{aligned}$$

使用并行替换规则，我们只需一步即可分别得到 w 的前缀和后缀：

$$\begin{aligned}
P = T[* \mapsto \texttt{22}] &= \texttt{1122112211},\\
S = T[* \mapsto \texttt{00}] &= \texttt{1100110011}.
\end{aligned}$$

我们将中间块和最终拼接构造为

$$\begin{aligned}
\texttt{000} &= \texttt{00} + \texttt{0},\\
\texttt{1122112211000} &= P + \texttt{000},\\
w &= \texttt{11221122110001100110011} = \texttt{1122112211000} + S.
\end{aligned}$$

因此，上述模板化组装方案可以写成逐步推导（每一行都是一次单位代价操作）：

$$\begin{aligned}
\texttt{11} &= \texttt{1} + \texttt{1},\\
\texttt{11*} &= \texttt{11} + *,\\
\texttt{11*11*} &= \texttt{11*} + \texttt{11*},\\
\texttt{11*11*11} &= \texttt{11*11*} + \texttt{11},\\
\texttt{22} &= \texttt{2} + \texttt{2},\\
\texttt{00} &= \texttt{0} + \texttt{0},\\
P &= \texttt{11*11*11}[* \mapsto \texttt{22}] = \texttt{1122112211},\\
S &= \texttt{11*11*11}[* \mapsto \texttt{00}] = \texttt{1100110011},\\
\texttt{000} &= \texttt{00} + \texttt{0},\\
\texttt{1122112211000} &= P + \texttt{000},\\
w &= \texttt{11221122110001100110011} = \texttt{1122112211000} + S.
\end{aligned}$$

按复合步骤计数，这个模板化组装方案的长度为 11：构建 T 需要四步，构建 22 和 00 需要两步，通过两次并行替换得到 P 和 S，构建 000 需要一步，最后还需要两次拼接。因此

$$\mathrm{TAI}(w) \le 11.$$

此外，对于这个实例，上面给出的规范计划长度为 12，且这个界是紧的，因此

$$\mathrm{ASI}(w) = 12.$$

因此，对于同一个目标字符串，我们得到严格分离：

$$\mathrm{TAI}(w) < \mathrm{ASI}(w).$$

3.3 另一个详细示例 I：分离 ASI 和 TAI

考虑字符串

$$w_1 = 101131211010112121111211,$$

我们给出另一个基于字母表 $\Sigma = \{0,1,2,3\}$ 的详细示例，以说明规范组装指数与模板化组装指数之间的严格分离。

规范组装计划（展示 ASI(w_1) = 13）。 在规范组装空间 $A(w_1)$ 中，我们将讨论范围限定为 w_1 的子串和二元拼接。对于 w_1，一种可行的组装计划如下（每行是一个单位代价步骤）：

$$\begin{aligned}
01 &= 0 + 1,\\
21 &= 2 + 1,\\
011 &= 01 + 1,\\
211 &= 21 + 1,\\
0113 &= 011 + 3,\\
1211 &= 1 + 211,\\
01121 &= 011 + 21,\\
10113 &= 1 + 0113,\\
121101 &= 1211 + 01,\\
2111211 &= 211 + 1211,\\
10113121101 &= 10113 + 121101,\\
011212111211 &= 01121 + 2111211,\\
101131211010112121111211 &= 10113121101 + 011212111211.
\end{aligned}$$

这得到了一个长度为 13 的标准组装计划。我们已通过计算验证，不存在长度不超过 12 的标准组装计划。因此

$$ASI(w_1) = 13.$$

模板 d 组装计划和 TAI(w_1)。 为便于阅读，将因子 w_1 划分为连续的块 ks

$$w_1 = 1011\ 3\ 1211\ 0\ 1011\ 2\ 1211\ 1211.$$

两个经常重复使用的块 1011 和 1211 共享同一个锚点骨架

$$1 \square 11,$$

其中，占位符 $\square$ 在 1011 中取值为 0，在 1211 中取值为 2。因此，在模板化组装空间 $A_*(w_1)$ 中，我们使用通配符 $*$ 和块压缩模板

$$T = 1 * 11.$$

逐步进行模板化推导（每一行都是一个单位代价操作）：

$$\begin{aligned}
11 &= 1 + 1,\\
1* &= 1 + *,\\
T = 1 * 11 &= 1 * +11,\\
1011 &= T[* \mapsto 0],\\
1211 &= T[* \mapsto 2],\\
10113 &= 1011 + 3,\\
101131211 &= 10113 + 1211,\\
1011312110 &= 101131211 + 0,\\
10113121101011 &= 1011312110 + 1011,\\
101131211010112 &= 10113121101011 + 2,\\
1011312110101121211 &= 101131211010112 + 1211,\\
w_1 = 10113121101011212111211 &= 1011312110101121211 + 1211.
\end{aligned}$$

计算步骤数，我们需要 3 步构造 T，2 个实例化步骤得到 1011 和 1211，再通过 7 次拼接组装出最终字，共计 12 步。因此

$$\mathrm{TAI}(w_1) \le 12.$$

有了 $\mathrm{ASI}(w_1) = 13$ ，这就实现了严格分离：

$$\mathrm{TAI}(w_1) < \mathrm{ASI}(w_1).$$

3.4 额外的计算示例 II：分离 ASI 和 TAI

考虑字符串

$$w_2 = 10113121101011212111211011,$$

我们得到类似的分离结果。

标准组装计划（显示 ASI(w_2) = 14）。一种可能的标准组装计划如下（每行代表一个单位代价步骤）：

$$
\begin{aligned}
01 &= 0 + 1,\\
10 &= 1 + 0,\\
11 &= 1 + 1,\\
12 &= 1 + 2,\\
0101 &= 01 + 01,\\
1011 &= 10 + 11,\\
1211 &= 12 + 11,\\
10113 &= 1011 + 3,\\
121211 &= 12 + 1211,\\
12110101 &= 1211 + 0101,\\
12111011 &= 1211 + 1011,\\
1011312110101 &= 10113 + 12110101,\\
12121112111011 &= 121211 + 12111011,\\
10113121101011212111211011 &= 1011312110101 + 12121112111011.
\end{aligned}
$$

这给出了一个长度为 14 的标准组装计划。我们已通过计算验证，不存在长度不超过 13 的标准组装计划。因此

$$ASI(w_2) = 14.$$

模板化组装方案和 $\mathrm{TAI}(w_2)$。因子

$$w_2 = 1011\ 3\ 1211\ 0\ 1011\ 2\ 1211\ 1211\ 1011.$$

与之前一样，使用模板 $T = 1 * 11$ 捕获的骨架 $1\square 11$。逐步的模板化推导如下：

$$
\begin{aligned}
11 &= 1 + 1,\\
1* &= 1 + *,\\
T = 1 * 11 &= 1 * +11,\\
1011 &= T[* \mapsto 0],\\
1211 &= T[* \mapsto 2],\\
10113 &= 1011 + 3,\\
101131211 &= 10113 + 1211,\\
1011312110 &= 101131211 + 0,\\
10113121101011 &= 1011312110 + 1011,\\
101131211010112 &= 10113121101011 + 2,\\
1011312110101121211 &= 101131211010112 + 1211,\\
10113121101011212111211 &= 1011312110101121211 + 1211,\\
w_2 = 10113121101011212111211011 &= 10113121101011212111211 + 1011.
\end{aligned}
$$

步骤计数：构建 T 需要 3 步，2 次实例化，以及 8 次拼接（对应 9 个块），总计 13 步。因此

$$\mathrm{TAI}(w_2) \le 13,$$

并且对于 ASI(w_2) = 14，我们再次得到

$$\mathrm{TAI}(w_2) < \mathrm{ASI}(w_2).$$

4 与文法和基于模式的形式体系的关系

在$\mathcal{A}_*(w)$中引入通配符模板，将模板化组装理论与理论计算机科学和形式语言理论中的既有形式体系联系起来。

首先，规范字符串组装指数与最小文法问题密切相关：给定字符串 w，找到一个能恰好生成 w 的最小上下文无关文法（或直线程序）。在 [5] 中，这种对应关系得到了精确阐述，并证明了计算 ASI(w) 的问题是 NP 完全的。

模板化组装将这一图景拓展至以下方向：宏语法和参数化语法，其中非终结符可以接受参数，从而有效地编码带空位的模板；模式语言，其中模式是由终结符和变量构成的字母表上的字符串，变量可以用任意字符串实例化；以及插入系统，其中字符串通过在指定位置插入子串而增长。

我们的块压缩模板对应于这样的模式：其字面片段受限于必须按照在 w 中出现的相同顺序排列，而其变量（通配符）则表示这些锚点之间可压缩的块。

因此，模板化组装指数 TAI(w) 可以视为一种操作性度量，用于衡量通过一类受限的基于模式的构造以多高的效率生成 w；这类构造将拼接与有限形式的参数化复制相结合。这一视角既启发了算法策略（例如，通过基于语法的压缩和宏规则来近似计算 TAI(w)），也引出了复杂性理论结果，下面我们将转而讨论这些结果。

4.1 用于近似$\mathrm{TAI}(w)$的贪心宏语法启发式方法

与宏语法的联系启发我们采用实用的启发式方法，为 TAI(w) 求得可计算上界。通俗地说，可以将一个可频繁实例化的模式视为一条“宏规则”，其规则体是模板 $T \in T(w)$（即骨架），而其参数是用于通过 R_{temp}实例化通配符位置的填充块 u。

候选模板和填充符族。 固定一个模板 $T \in T(w)$。对于每个 $u \in O^*$，若完全实例化后的字符串

$$x_u := T[P(T) \mapsto u]$$

是 w 的子串（因此 $x_u \in \mathrm{Sub}(w) \subseteq O^*$），令 occ($x_u$) 表示$x_u$ 在 w 中出现的次数（或在启发式方法维护的 w 当前工作表示中的出现次数）。与其选择单个二元组 (T, u)，不如选择一个骨架 T，以及一个满足 $x_u \in \mathrm{Sub}(w)$ 的填充符族 $U \subseteq O^*$。这反映出，构造骨架 T（通过 R_{concat}）的成本可以分摊到多次实例化中，而这些实例化可能使用不同的填充符。实际上，通常最好限制为至少有两个通配符位置的模板，$|P(T)| \geq 2$，因为这样一次实例化步骤便可同时填充多个位置（选择 $S = P(T)$）。

一个简单的收益评分。为判断采用一个骨架 T 和一个填充族U 是否可能减少步骤数，我们比较：(i) 在规范模型中“直接”组装所有实例化子串 x_u 的出现次数，以及 (ii) 通过 R_{concat} 组装一次骨架 T，将每个填充符 $u \in U$ 各组装一次，然后在 R_{temp}中通过一次实例化生成 x_u 的每次出现。令 $c(\cdot)$ 表示任意一种可高效计算的成本代理，用于衡量规范（仅拼接）组装复杂性，其目的是通过考虑经典复用来近似 ASI($\cdot$)（或为其提供上界）。在最简单的基准方案中，可以取 $c(y) := |y| - 1$。利用这样的代理，定义

$$\mathrm{gain}(T, U) := \sum_{u \in U} \mathrm{occ}(x_u) \cdot c(x_u) - \Big(c(T) + \sum_{u \in U} c(u) + \sum_{u \in U} \mathrm{occ}(x_u) \Big), \qquad x_u := T[P(T) \mapsto u].$$

正分数表示（在此代理指标下），值得付出代价来构造 T（一次）以及 U 中的填充符，以便将成本分摊到各次实例化上。在贪心实现中，通常只接受满足 gain$(T,U) > 0$ 的候选项，并将 gain(T,U) 的大小用作优先级分数（收益越大，表示该模板在所选代理指标下越有希望）。反之，gain$(T,U) < 0$ 表明该模板在此近似下可能无益，不过不能排除它在完整 TAI模型中通过后续复用效应带来间接收益。

贪心启发式概述。 从显式目标词 w开始，迭代 e:

1. 挖掘候选骨架 $T \in T(w)$及其可采纳填充符 u，使得 $x_u = T[P(T) \mapsto u] \in \mathrm{Sub}(w)$，并计算当前工作表示中 occ$(x_u)$。 2. 对每个 T，选择一个填充族 U（例如所有可采纳填充符，或得分最高的子集），并计算 gain$(T,\ U)$。 3. 选择具有最大正增益的骨架 T 及其填充族 U。 4. 确定只构造一次 T（使用 R_{concat}），每个 $u \in U$ 只构造一次，并通过 R_{temp} 中的单次规则应用（完全并行替换，$S = P(T)$）生成 x_u 的每个选定出现位置。 5. 将每个x_u 的每个生成出现位置视为可用对象，以此“压缩” w 的工作表示，并继续在剩余结构中搜索更有收益的候选项。

由此得到的已提交构造和实例化结果可以直接转换为 $A^*(w)$ 中的模板化组装方案，因此为 TAI(w) 给出一个明确的可计算上界。我们将此类启发式方法的详细实验评估留待未来工作。

这一观点与例3.2一致：骨架 $T = \texttt{11*11*11}$ 在 w中通过两个不同的填充符进行实例化，即 $u = \texttt{22}$和 $u = \texttt{00}$，从而生成 $P = T[P(T) \mapsto \texttt{22}]$和$S = T[P(T) \mapsto \texttt{00}]$。对于基线选择 $c(y) = |y| - 1$，我们有 $|T| = 8$，因此 $c(T) = 7$；而 $|\texttt{22}| = |\texttt{00}| = 2$，所以 $c(\texttt{22}) = c(\texttt{00}) = 1$。此外，$|P| = |S| = 10$，因此 $c(P) = c(S) = 9$，并且在此例中，occ$(P) =$ occ$(S) = 1$。取 $U = \{\texttt{22},\ \texttt{00}\}$后，聚合分数变为

$$\mathrm{gain}(T,U) = (1 \cdot 9 + 1 \cdot 9) - \big(7 + (1+1) + (1+1)\big) = 18 - 11 = 7,$$

其为正，恰恰是因为构造骨架 T 的一次性成本由使用不同填充符的两次实例化分摊。因此，基于增益的贪心策略自然会选择只组装一次单个骨架 T，然后针对每种填充符分别实例化，从而实现该示例中模板化组装方案所体现的步数节省。

paragraph增益计算适用于另外两个详细示例 I 和 II。我们沿用基线代理 $c(y) := |y| - 1$。

示例 I ($w_1 = 10113121101011212111211$)。取骨架 $T = 1 * 11$ 和填充族$U = \{\texttt{0},\ \texttt{2}\}$，由此得到

$$x_0 = T[* \mapsto 0] = 1011, \qquad x_2 = T[* \mapsto 2] = 1211.$$

此处 $|T| = 4$，故 $c(T) = 3$，且 $|0| = |2| = 1$，从而 $c(\texttt{0}) = c(\texttt{2}) = 0$。此外 $|x_0| = |x_2| = 4$，所以 $c(x_0) = c(x_2) = 3$。在块分解中

$$w_1 = 1011\ 3\ 1211\ 0\ 1011\ 2\ 1211\ 1211$$

我们有 $\mathrm{occ}(x_0) = 2$ 且 $\mathrm{occ}(x_2) = 3$。因此

$$\begin{aligned}\mathrm{gain}(T, U) &= \big(2 \cdot c(x_0) + 3 \cdot c(x_2)\big) - \Big(c(T) + \big(c(0) + c(2)\big) + \big(\mathrm{occ}(x_0) + \mathrm{occ}(x_2)\big)\Big) \\ &= (2 \cdot 3 + 3 \cdot 3) - (3 + (0 + 0) + (2 + 3)) \\ &= 15 - 8 = 7.\end{aligned}$$

示例 II ($w_2 = 10113121101011212111211 1011$)。使用相同的 $T = 1 * 11$ 和 $U = \{0,2\}$，我们仍有 $c(T) = 3$, $c(0) = c(2) = 0$，以及 $c(1011) = c(1211) = 3$。使用该因式分解

$$w_2 = 1011\ 3\ 1211\ 0\ 1011\ 2\ 1211\ 1211\ 1011,$$

我们得到 $\mathrm{occ}(1011) = 3$ 和 $\mathrm{occ}(1211) = 3$，因此

$$\begin{aligned}\mathrm{gain}(T, U) &= (3 \cdot 3 + 3 \cdot 3) - (3 + (0 + 0) + (3 + 3)) \\ &= 18 - 9 = 9.\end{aligned}$$

5 计算复杂性

我们考虑与模板化组装指数 $\mathrm{TAI}(w)$ 相关联的判定问题。

定义 5.1。 判定问题 TAI-DEC 定义如下：

- 实例：一个字符串 $w \in \Sigma^+$ 以及一个非负整数 k （以一元或二进制形式给出）。
- 问题：在模板化组装空间$\mathcal{A}_*(w)$中，是否存在针对 w 、总成本至多为 k 的模板化组装方案?

引理 5.2. *TAI-DEC* 属于NP。

证明概要。 证书是一系列规则应用（拼接与模板实例化），以及它们所产生的中间对象。由于每一步都会严格增加至少一个已构造对象的长度，有效计划的长度关于 $|w| + k$多项式有界。验证通过模拟每一步操作，并检查所产生的对象属于 $O_* = \mathrm{Sub}(w) \cup \mathcal{T}(w)$来进行。所有检查均可在关于$|w| + k$的多项式时间内完成，因此 TAI-DEC 属于 NP。

□

关于难性。 规范判定问题

$$\text{ASI-DEC} = \{\, (w, k) \mid \mathrm{ASI}(w) \le k \,\}$$

对于固定的有限字母表 [5]是 NP 完全的。由于对所有 w都有 $\mathrm{TAI}(w) \le \mathrm{ASI}(w)$，模板化模型是规范模型的真正扩展。然而，这一扩展朝着更容易的方向进行（它可能缩短最优计划），因此 TAI-DEC 的 NP 难性并不能直接由 ASI-DEC 的 NP 难性得出。

不过，仍可将 ASI-DEC 视为 TAI-DEC 在从不应用 R_{temp} 中规则的组装计划上的限制。等价地，在承诺 w 不允许有益的模板实例化（因此 $\mathrm{TAI}(w) = \mathrm{ASI}(w)$）的条件下，问题 TAI-DEC 与 ASI-DEC 是一致的。

然而，这一观察并未解决 TAI-DEC 的无条件最坏情况复杂度。事实上，一旦允许有益的模板实例化，判断 $\mathrm{TAI}(w) \le k$ 就必须同时考量经典的复用与串联权衡，以及是否值得投入串联步骤来构建模板，以便通过一个或多个实例化步骤分摊这些成本。因此，我们强烈猜想

TAI-DEC 仍然是 NP 困难的（即使在固定字母表上也是如此）：模板化模型保留了规范组装规划的组合“硬核”，同时还增加了一层离散结构选择（识别并利用有利模板）。虽然这并非证明，但它有力地表明，最坏情况下的难解性应当依然存在。对于固定字母表，确立 TAI-DEC 的无条件 NP 难性（或多项式时间可解性）仍是一个值得探讨的开放问题。

6 讨论

我们提出了对组装理论规范的基于字符串的表述的一种扩展，其中中间对象可以包含表示压缩块的通配符 $*$，并且模板被限制为目标字符串的块压缩子串。由此得到模板化组装指数 TAI(w)，这是一种新的复杂度度量，它推广了规范组装指数 ASI(w)，并对模板化模块性敏感：即在由共同支架连接的多个、可能相距较远的上下文中反复出现的模块。

形式上，我们定义了模板化组装空间，规定了拼接与模板实例化规则，并建立了 TAI(w) 与 ASI(w) 之间的基本关系。我们确立了其属于 NP，并阐明了为何 NP 难性并不能由一个平凡的嵌入论证得出。我们猜想 TAI-DEC 是 NP 困难的（即便对于固定字母表亦然），同时将无条件的 NP 难性留作开放问题。我们还概述了其与基于语法的压缩以及模式语言等形式体系之间的联系。

TAI 及其未来变体可能提供新见解的一些领域包括：

- 化学与分子设计：许多分子都可以描述为相对刚性的支架，上面装饰着重复的官能团：带有多个相同取代基的芳香核心、在多个位置连接相同侧链的低聚物，或由少量分支单元构建而成的树枝状结构。在分子的基于字符串的表示中（例如，用线性化编码表示带有取代基的核心），这类模式对应于模板，其中通配符位置标记了官能化位点。规范组装指数会量化重复片段的复用，但无法区分彼此独立的重复与由共享骨架协调的重复。相比之下，模板化组装明确体现了这一区别：首先构建一个带有通配符位置的共同支架，然后用选定的官能团实例化。ASI(w) –TAI(w) 差距较大的分子或分子系列，便对应于这样一类结构：其表面上的大部分组合复杂性源自模板化取代模式，而非真正互不相关的子基序。这表明，该方法可用于量化药物化学、组合文库设计以及聚合物或超分子工程中的支架—装饰模块性。

- 序列分析与基因组学：许多基因组区域呈现模块化结构，其中反复出现的基序（例如转录因子结合位点、蛋白质结构域）嵌入于不同的局部上下文中。传统组装指数已能衡量层级复用，但模板化指数或许能更直接地反映共享支架中同一模体的多次出现。ASI(w) 和 TAI(w) 在不同基因组区域中的差异表现，或许能凸显具有功能意义的模板化结构。

- 生物特征与生命起源：有人提出，组装理论可作为一种框架，通过基于假定的组装速率推导出的复杂性度量来定义普适生物特征 [1, 2, 3]。模板化组装指数提供了一个互补的维度：较高的

TAI(w)与较大的差距 ASI(w) − TAI(w)可能表明某些系统以体现进化生物组织特征的方式利用模块的模板化复制，例如处于共同调控架构下的基因家族。

未来的研究可以包括：针对不同字符串族，对差值 ASI(w) − TAI(w) 进行更详细的分类；完善复杂性与近似结果；以及在生物和人工数据集上对 TAI 开展实证研究。在概念层面，模板化组装理论提出了一个更广泛的研究纲领：系统探索不同类别的模板和重写操作如何产生不同的组装指数族，而每一类指数都能探究复杂系统中结构与历史的不同侧面。

参考文献

[1] Marshall SM、Murray ARG、Cronin L. 利用路径复杂性识别生物特征的概率框架。英国皇家学会哲学汇刊A辑：数学、物理与工程科学。2017年12月；375(2109)：20160342。获取地址： https://royalsocietypublishing.org/doi/10.1098/rsta.2016.0342。[2] Marshall SM、Moore DG、Murray ARG、Walker SI、Cronin L. 利用组装空间形式化生命起源路径。熵。2022年6月；24(7)：884。获取地址： https://www.mdpi.com/1099-4300/24/7/884。[3] Sharma A、Czégel D、Lachmann M、Kempes CP、Walker SI、Cronin L. 组装理论解释并量化选择与进化。《自然》。2023年10月；622(7982)：321-8。获取地址： https://www.nature.com/articles/s41586-023-06600-9。[4] Łukaszyk S、Bieniawski W. 二进制消息组装理论。《数学》。2024年5月；12(10)：1600。获取地址： https://www.mdpi.com/2227-7390/12/10/1600。[5] Masierak P. 确定组装指数的计算复杂性。《IPI快报》。2026年1月：9-12。获取地址： https://ipipublishing.org/index.php/ipil/article/view/315。